TrueSync Developer's Guide

Section 3: Interface Method Descriptions

Introduction

This section of the Developers Guide discusses in detail the interface methods that a programmer needs to satisfy when implementing an accessor or conduit.

Note that the `SyncCore.dll` implementation of the `SyncCore` logic only requires the use of the `SyncCore\IStore` and `ConduitKit\IInserter` interfaces. The `AccessKit\IAccessStore` interface used inbound between the conduit and the accessors is optional and assumes the use of data storage classes from `ToolKit`. This is also true of the implementation base class `ConduitKit\Transformer`.

Note that during the outbound phase, the `Store` is the one and only interface object that is instantiated.

Interface List

IStore *(outbound & inbound)*

`void destroy()`

A request to delete the object. Usually implemented inline as `{ delete this; }`.

`int open(ISyncInfo& info, int mode)`
`int close()`

`Open(...)` is a request to open the dataset and the record map. Configuration information may be retrieved from `ISyncInfo`. The dataset mode is `READ` for outbound processing and `WRITE` for inbound processing.

`Close()` is a request to close the dataset and the record map.

Return non-zero for success and zero for failure.

`IRecordMap& map()`

A request for the record map. Your store must instantiate a class using the `AccessKit\TRecordMap` template. See the header file for details.

`int moveRecordToFromMap()`

An instruction to locate in the dataset the record that corresponds to the "current" record in the `RecordMap`. Called outbound during delete order generation and inbound for both deletes and exports. The current record in the `RecordMap` is retrieved with the method `MapEntryExt& TRecordMap::current()`.

Return non-zero for success and zero for failure. If the record is located, it should be made the "current" record in the `Store`. This is usually accomplished by keeping a copy of the record in memory.

`time_t recordModifyTime()`

A query for the `Store`'s "current" record last modified timestamp.

IStore *(outbound only)*

`int extractByDate(time_t t)`

A request to prepare for outbound export record generation. The parameter is the time of the last outbound synchronization with the target dataset. This method allows large datasets to limit the number of records extracted for processing. After this call the extracted recordset is assumed ready for iteration. Return non-zero for success and zero for failure.

`int isEmpty()`
`int isEOF ()`

Queries about the state of extraction iteration. Called during export generation. `isEmpty()` should return non-zero if there are records to iterate over. `isEOF()` should return non-zero if iteration has traversed all records in the extracted set. The `OutSynchronizer` logic does not require multiple or reverse iteration of the extract set.

`void moveRecordFirst()`
`void moveRecordNext ()`

Instructions to iterate through the extracted records. Called during export generation. The record moved to becomes the "current" record in the `Store`.

`MapEntryExt* moveEntryToFromStore()`

An instruction to locate in the RecordMap the entry that corresponds to the "current" record in the Store. Called during export generation. Use the call TRecordMap::moveToInt(MAP_ENTRY_INT&). Return the located entry or NULL if not found.

MapEntryExt& addToMapForExport()

An instruction to add an entry into the RecordMap containing the Store's current record internal identifier and timestamp. Called during export generation. Use the call TRecordMap::addExport(MAP_ENTRY_INT&, const CTime&). Return the new map entry.

int filterForExport()

The store may perform filtering to block certain records from being exported. Called during export generation. Return non-zero to pass the record and zero to block it.

void sendSchema(CArchive& ar)

A request to serialize the Store's schema into the passed MFC archive. This request may be ignored if the schema is fixed, like Intera. Called during export generation, before record serialization.

The format is completely at the discretion of the Store, although the ToolKit\FieldDirectory class is available for describing an arbitrary schema.

void sendRecord(CArchive& ar)

A request to serialize the Store's "current" record into the passed MFC archive. The format is completely at the discretion of the Store, although the ToolKit\FieldDirectory class is available for describing an arbitrary data structure. Called during export generation.

Return to Interface List

IStore Interface *(inbound only)*

MapEntryExt& addToMapForImport(MapEntryExt& entryExt, time_t timestamp)

An instruction to add an entry into the RecordMap containing the Store's current record internal identifier and the passed MapEntryExt and timestamp. Called after a record is inserted into the Store. Use the call TRecordMap::addImport(MAP_ENTRY_INT&, MapEntryExt&, time_t). Return the new map entry.

void removeRecord()

Instruction to remove the `Store` processing.

Return to

IAccessStore Interface *(source & target)*

A request to delete the object. Usually implemented inline as `{ delete this; }`

(source only)

ITransHalf& recvSchema(CArchive& ar)

`Store`'s schema out of the passed MFC archive. The serialization request may be ignored if the schema was not serialized outbound. Called if the schema is found while `SyncSet`.

`ToolKit\TransHalf` object describing the schema. A consists of a last-modified timestamp and two , one for the schema and one for the "current" `TransHalf` and header files.

If you want to use the interface you need to become familiar with the building blocks of , namely `Fields` `FieldArrays`.

A request to create or overwrite the `Store` passed MFC archive. Called if a record is found while reading the .

Return to

IAccessStore Interface *(target only)*

ITransHalf& getSchema()

void fetchTgtFields()

void acceptRecord(const ITransHalf& th)

int filterForDelete()

int filterForInsert()

time_t insertRecord()

time_t updateRecord()